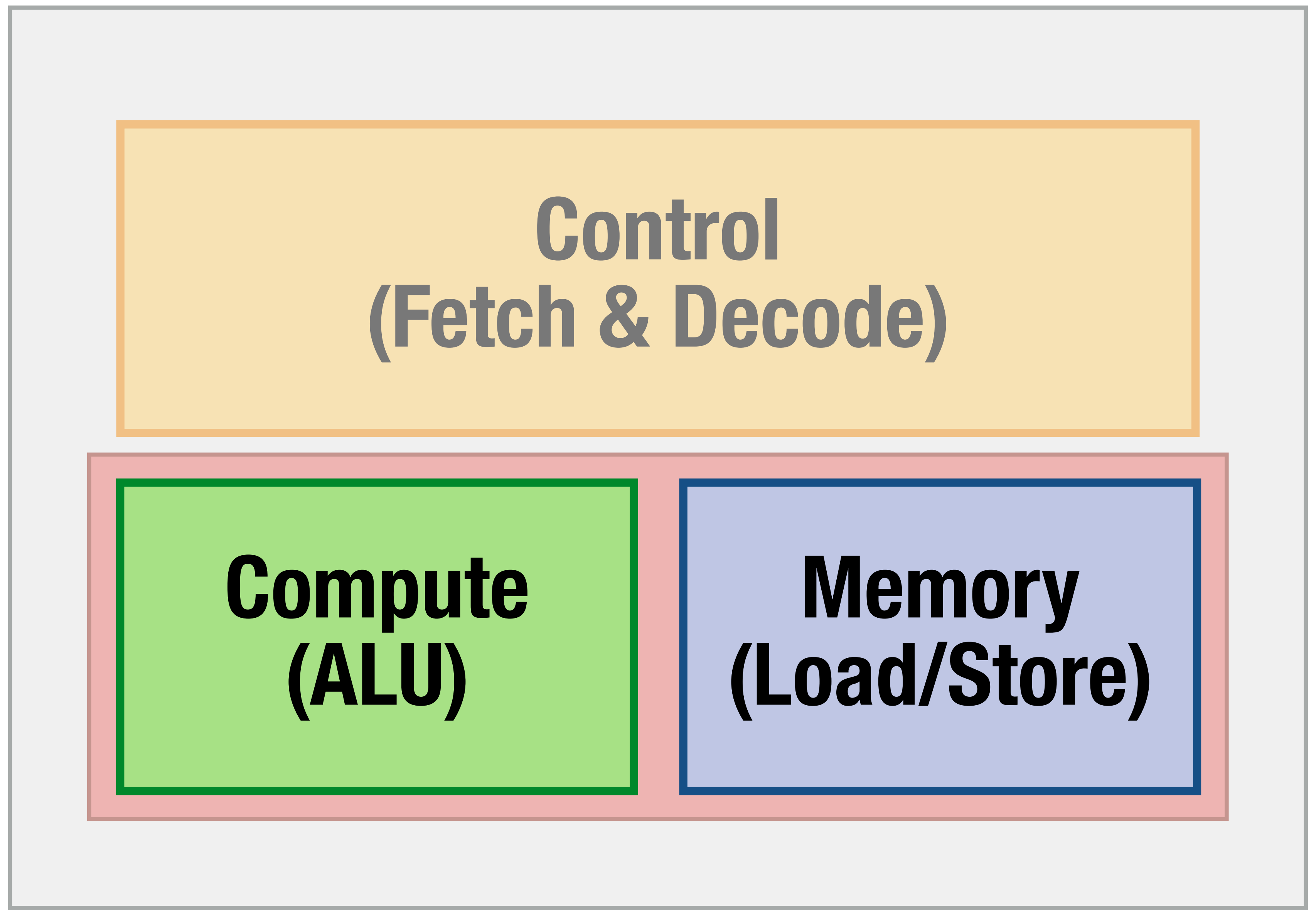
October 7, 2025

6.S894
Accelerated Computing

Live Lab 5:
Matmul, part 2

Jonathan Ragan-Kelley

Overlapping compute & I/O



Goal: fully utilize both resources

Problem 1: load / store instructions are **asynchronous & long-latency**

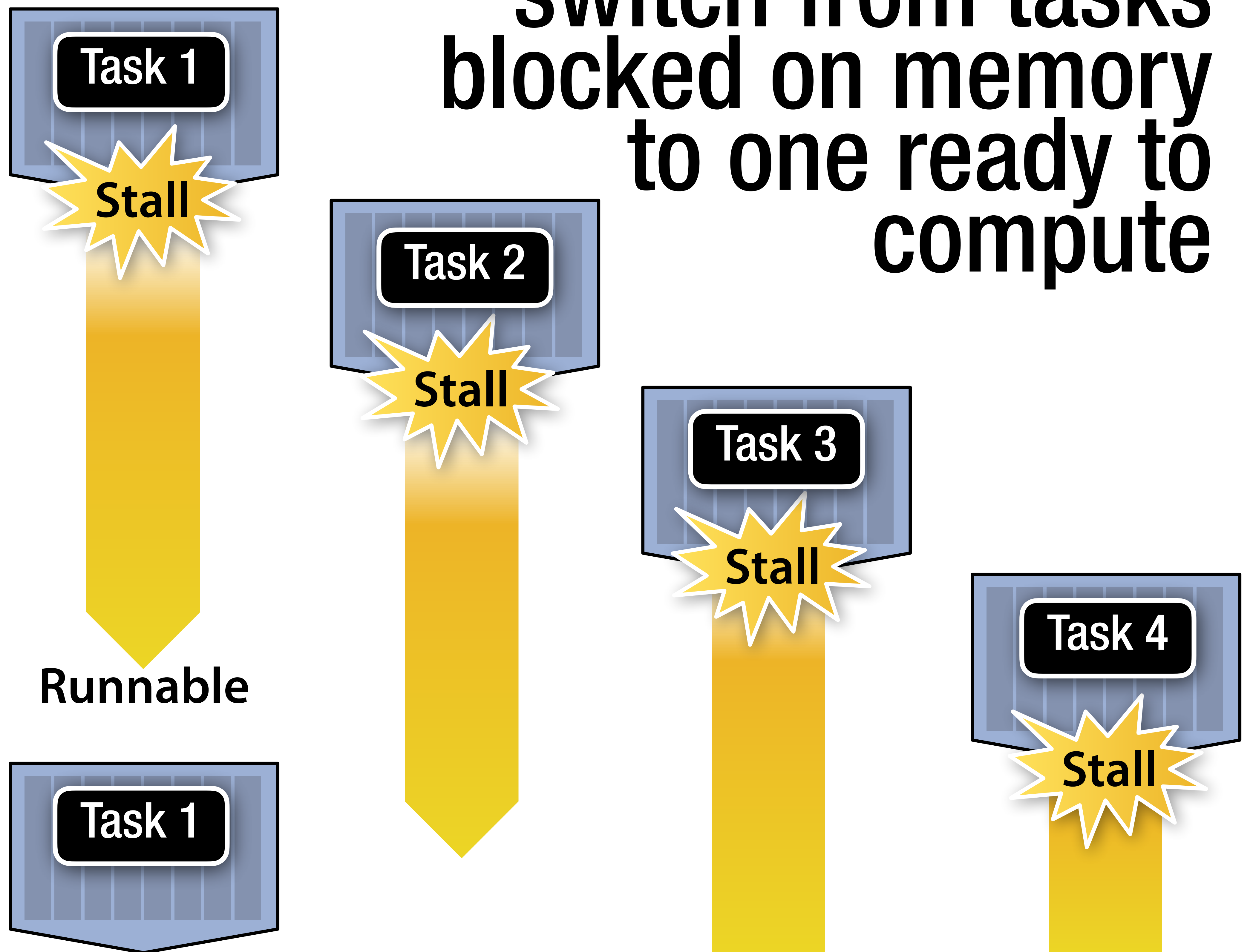
Solution 1: ILP

hoist loads early
to avoid blocking

```
ld         ld
fma        ld
ld    →    ld
fma        fma
ld         fma
fma        fma
…          …
```

Solution 2: multithreading

switch from tasks
blocked on memory
to one ready to
compute



Problem 2: load / store instructions waste **issue slots**

Solution: bulk load / store instructions
e.g.,“vectorized” ld / st

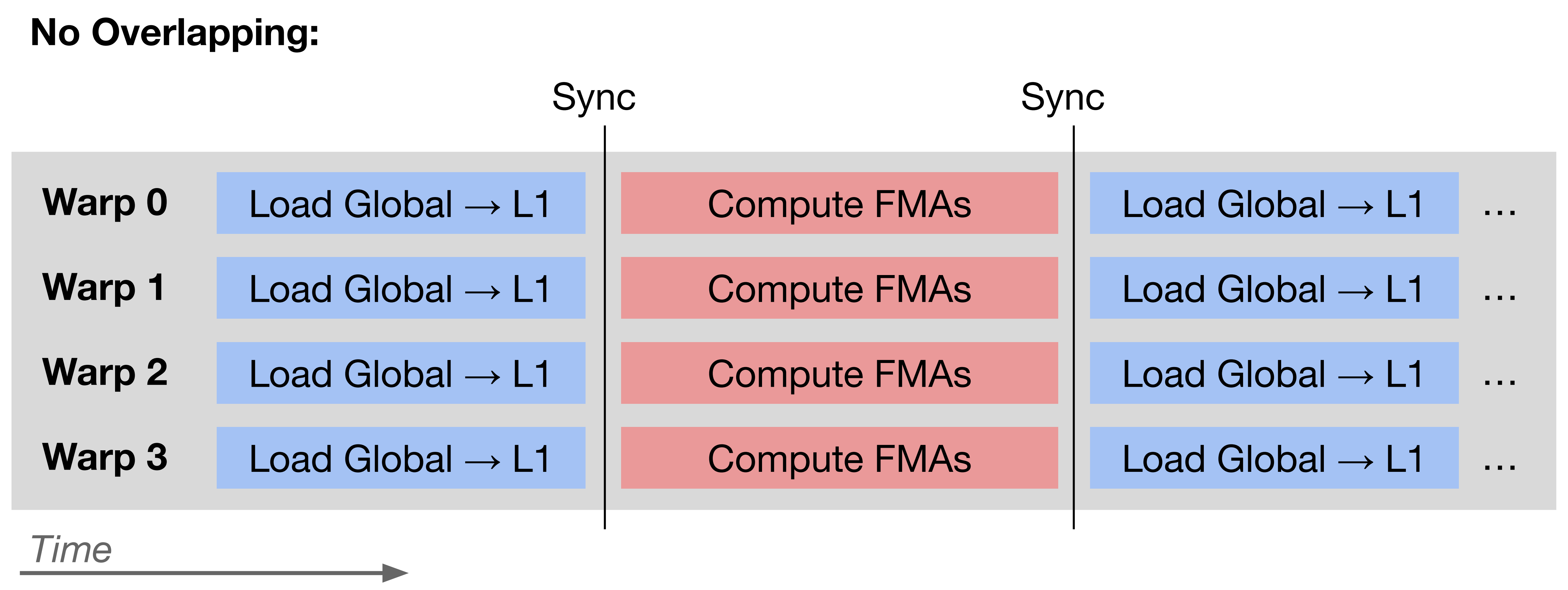
```
ld.f32
ld.f32
ld.f32
ld.f32
fma
fma
fma
fma
…
```

$\rightarrow$

```
ld.v4.f32
fma
fma
fma
fma
…
```

Problem 3: overlapping compute with loading to the **scratchpad**

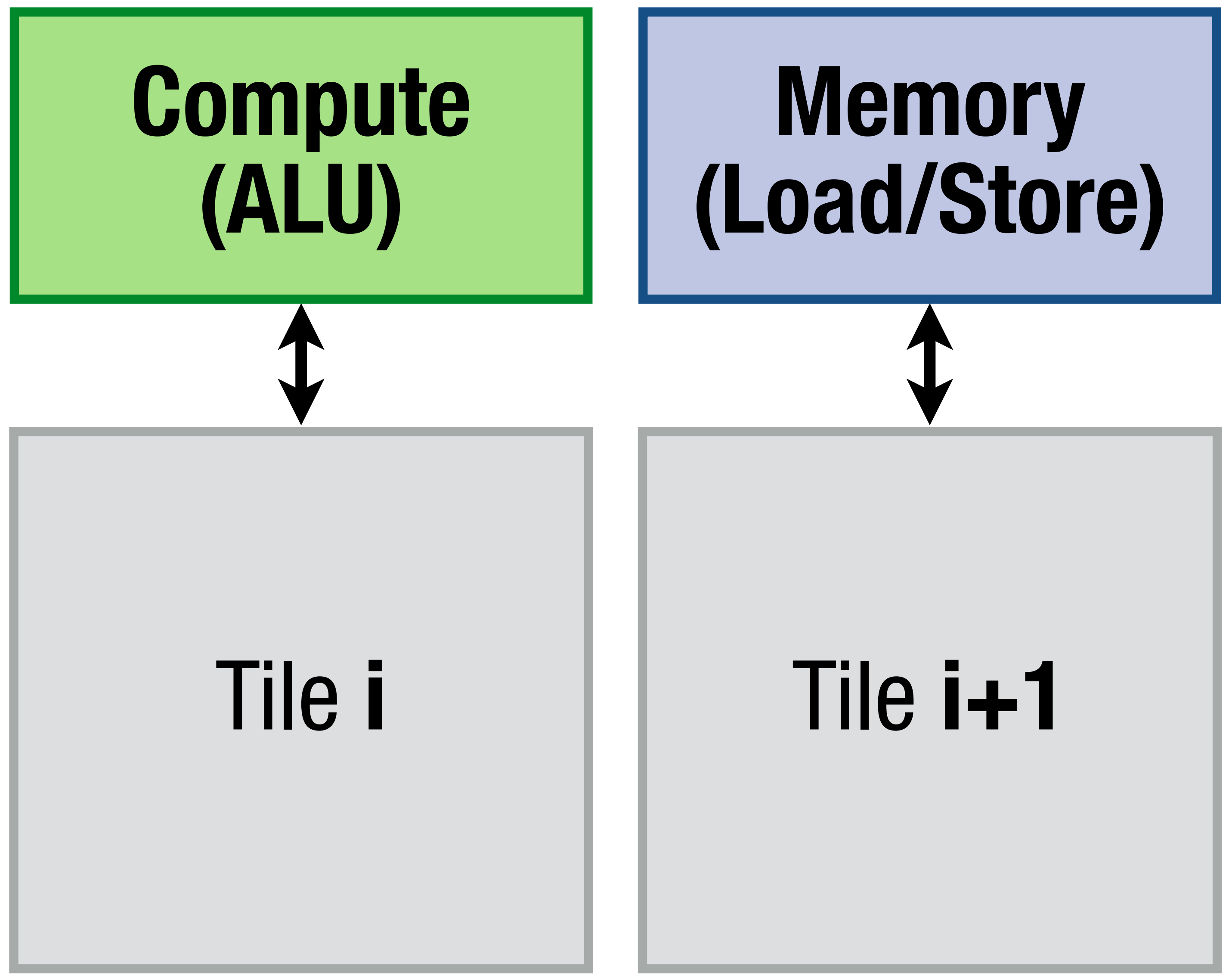
```
foreach tile:
  // load into scratchpad
  for i,j:
    load next A,B → scratch
  sync
  // compute!
  for i,j,k:
    compute C += A*B
  sync
```



Problem 3: overlapping compute with loading to the **scratchpad**

```
foreach tile:
  // load into scratchpad
  for i,j:
    load next A,B → scratch
  sync
  // compute!
  for i,j,k:
    compute C += A*B
  sync
```

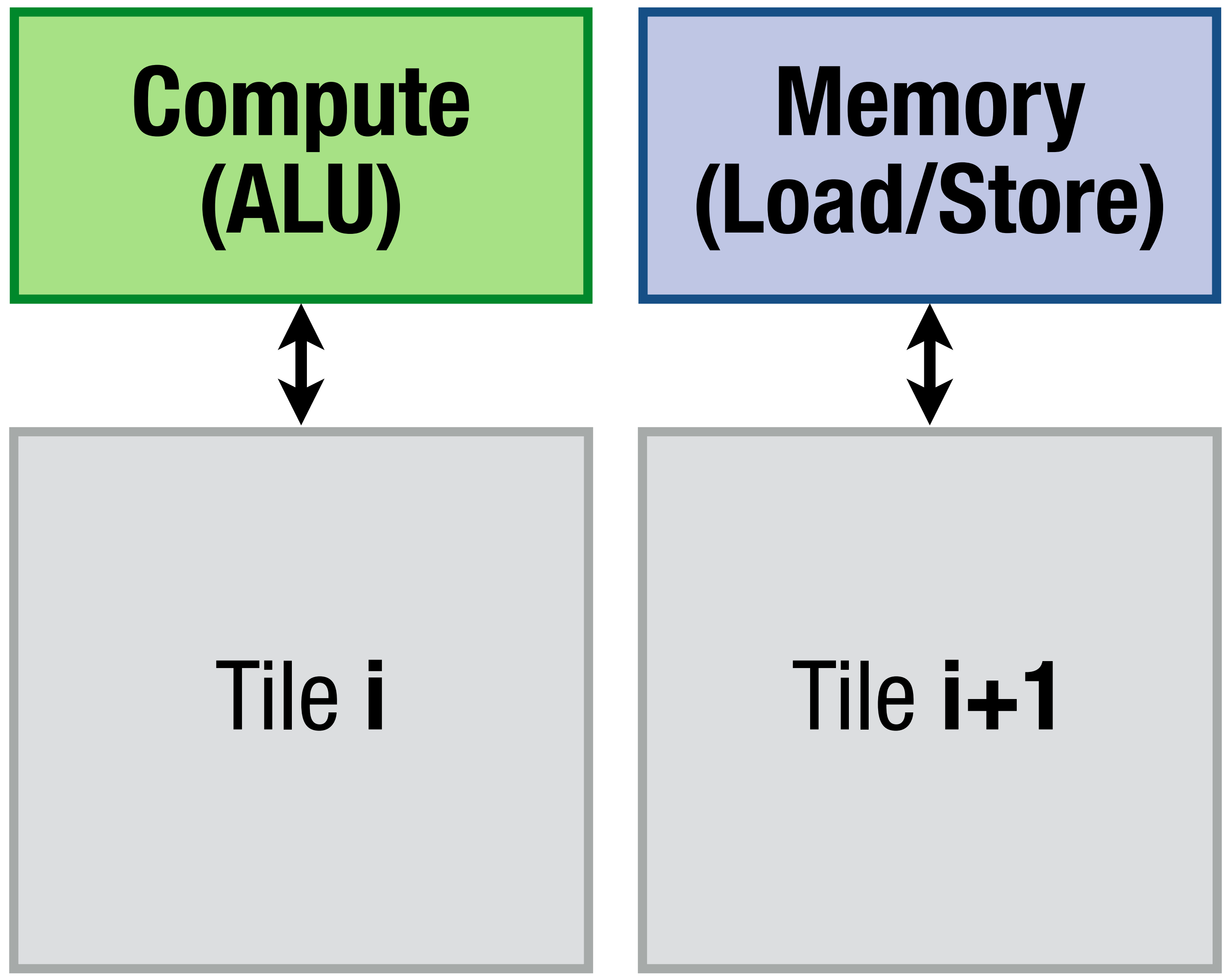
Solution:
asynchronous fetch &
double-buffering



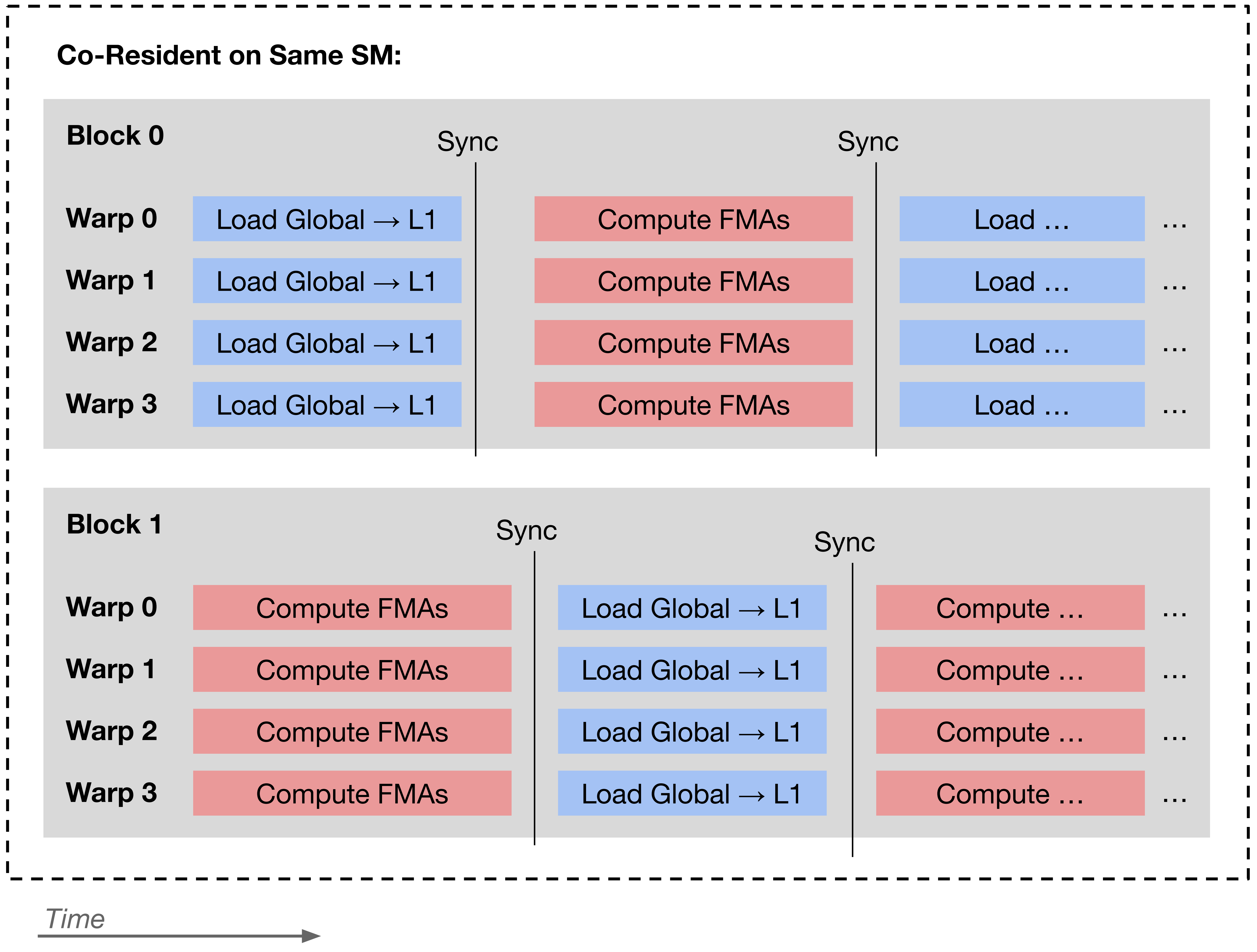
Problem 3: overlapping compute with loading to the **scratchpad**

```
foreach tile:
  // load into scratchpad
  for i,j:
    load next A,B → scratch
  sync
  // compute!
  for i,j,k:
    compute C += A*B
  sync
```

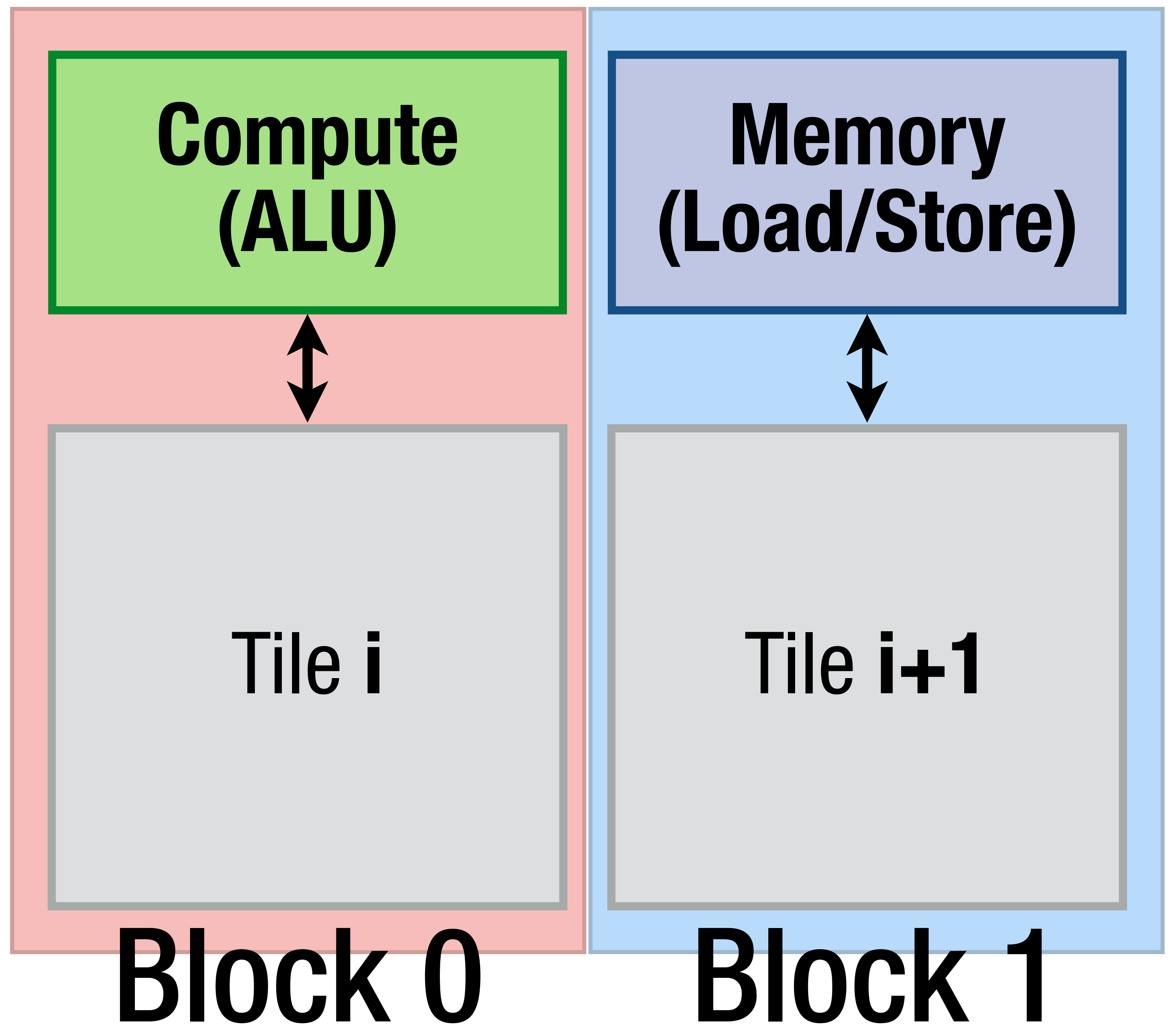
Solution:
asynchronous fetch &
double-buffering



Problem 3: overlapping compute with loading to the **scratchpad**



Solution: asynchronous fetch & **double-buffering**



Problem 3: overlapping compute with loading to the **scratchpad**

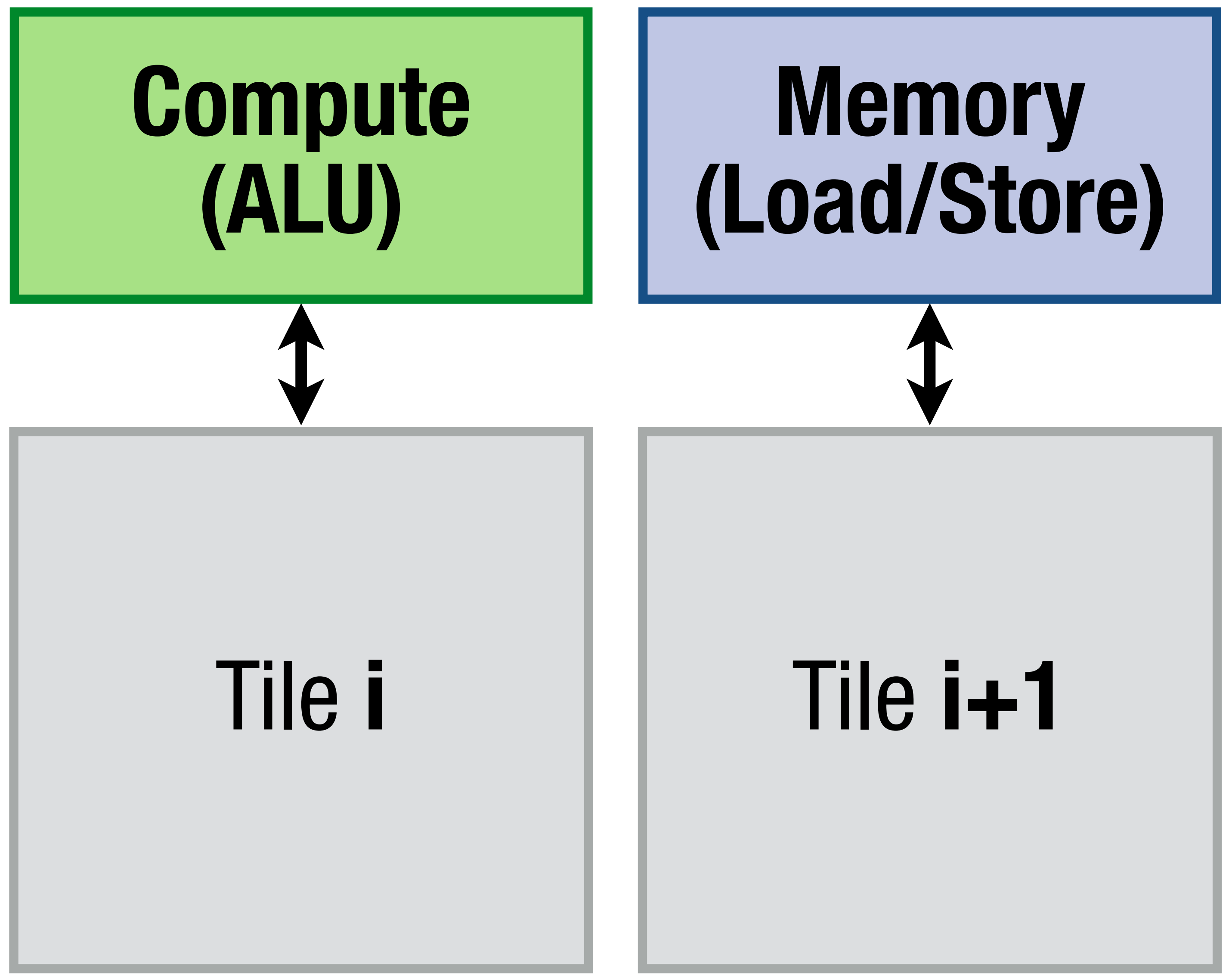
Implementation approach:

warp specialization

```
if threadIdx.y < 4:
  // load into scratchpad
  for i,j:
    load next A,B → scratch
else:
  // compute!
  for i,j,k:
    compute C += A*B
sync & swap buffers…
```

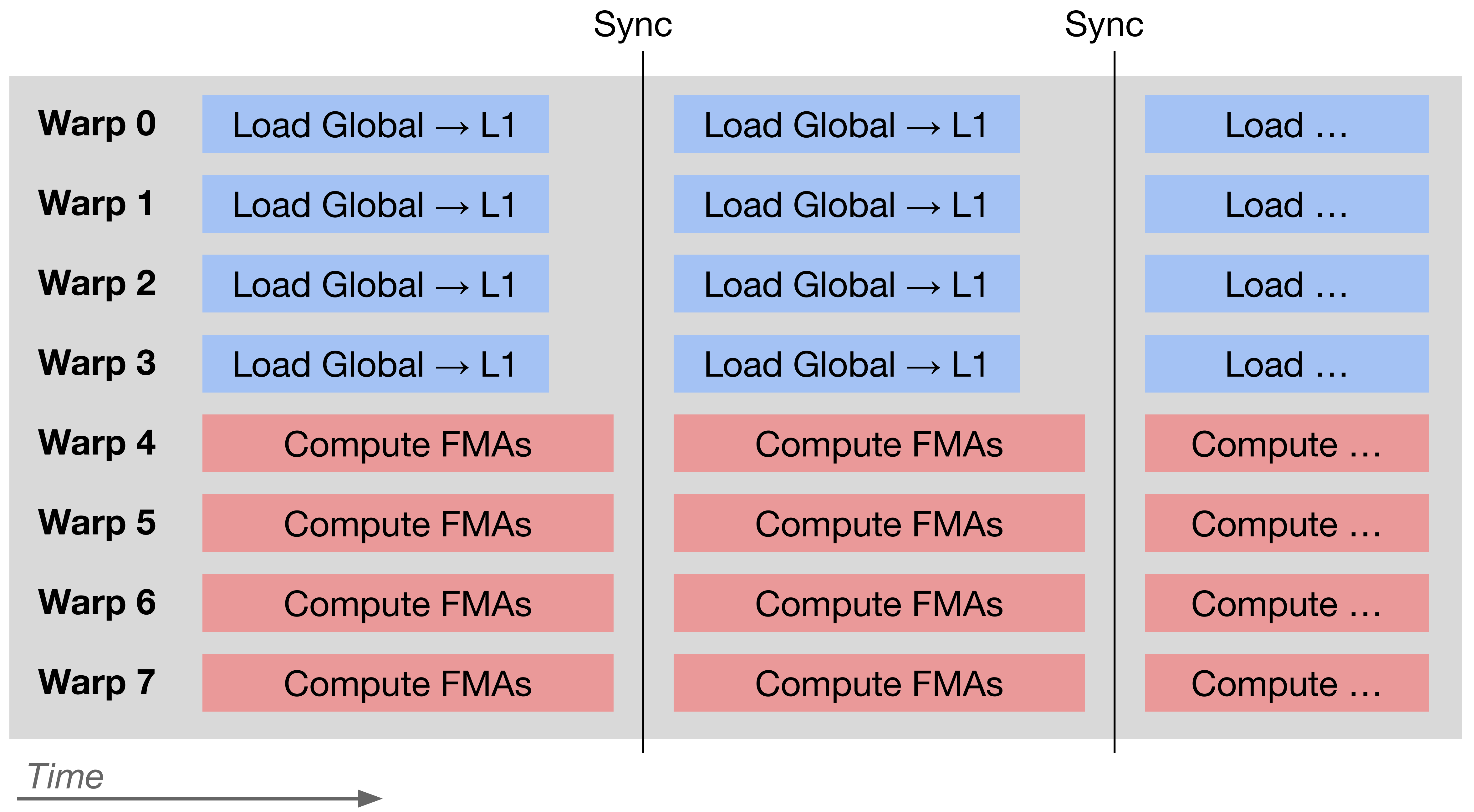
Solution:

asynchronous fetch & **double-buffering**

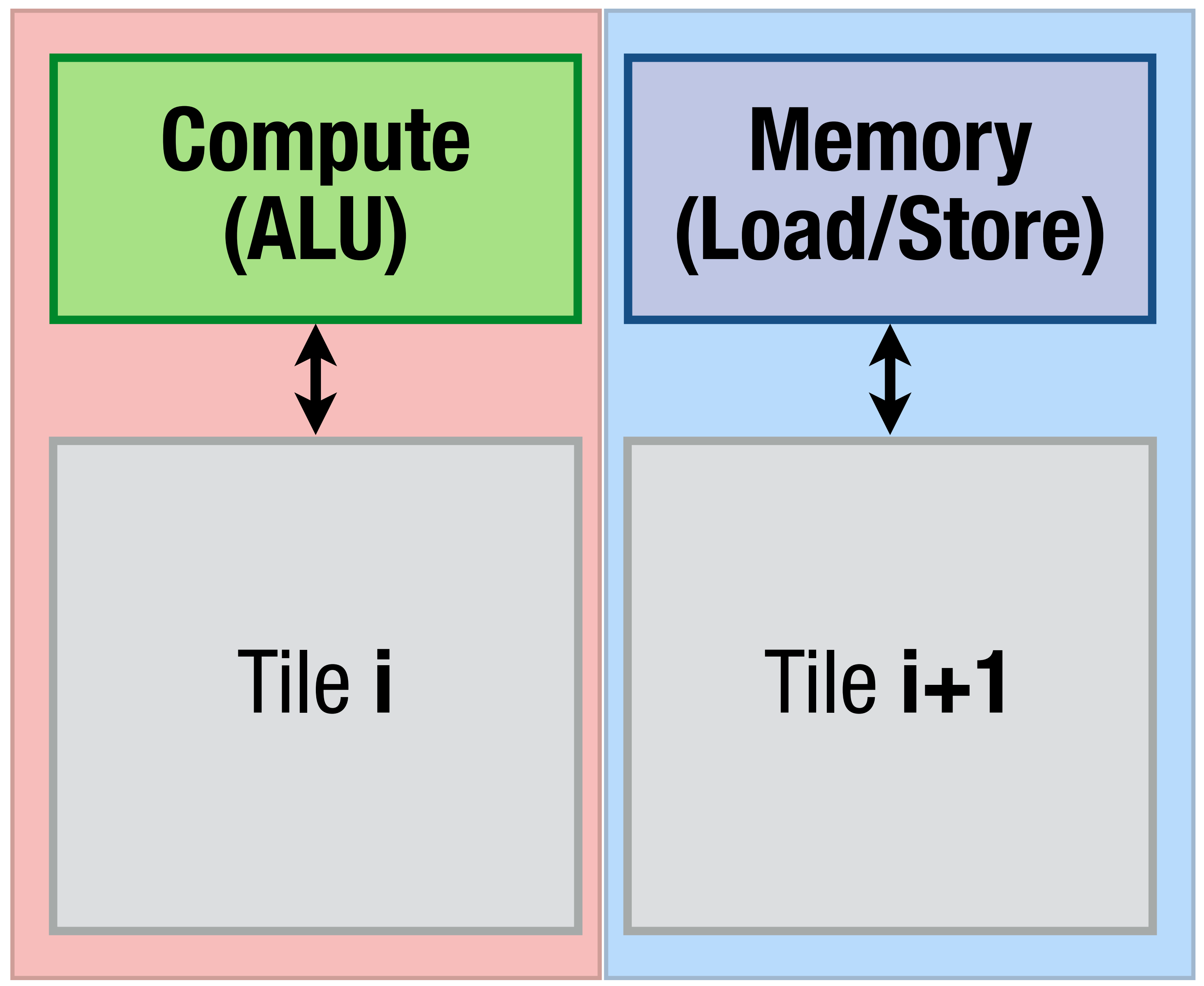


Problem 3: overlapping compute with loading to the **scratchpad**

Warp-Specialized Overlapping:



Solution: asynchronous fetch & **double-buffering**

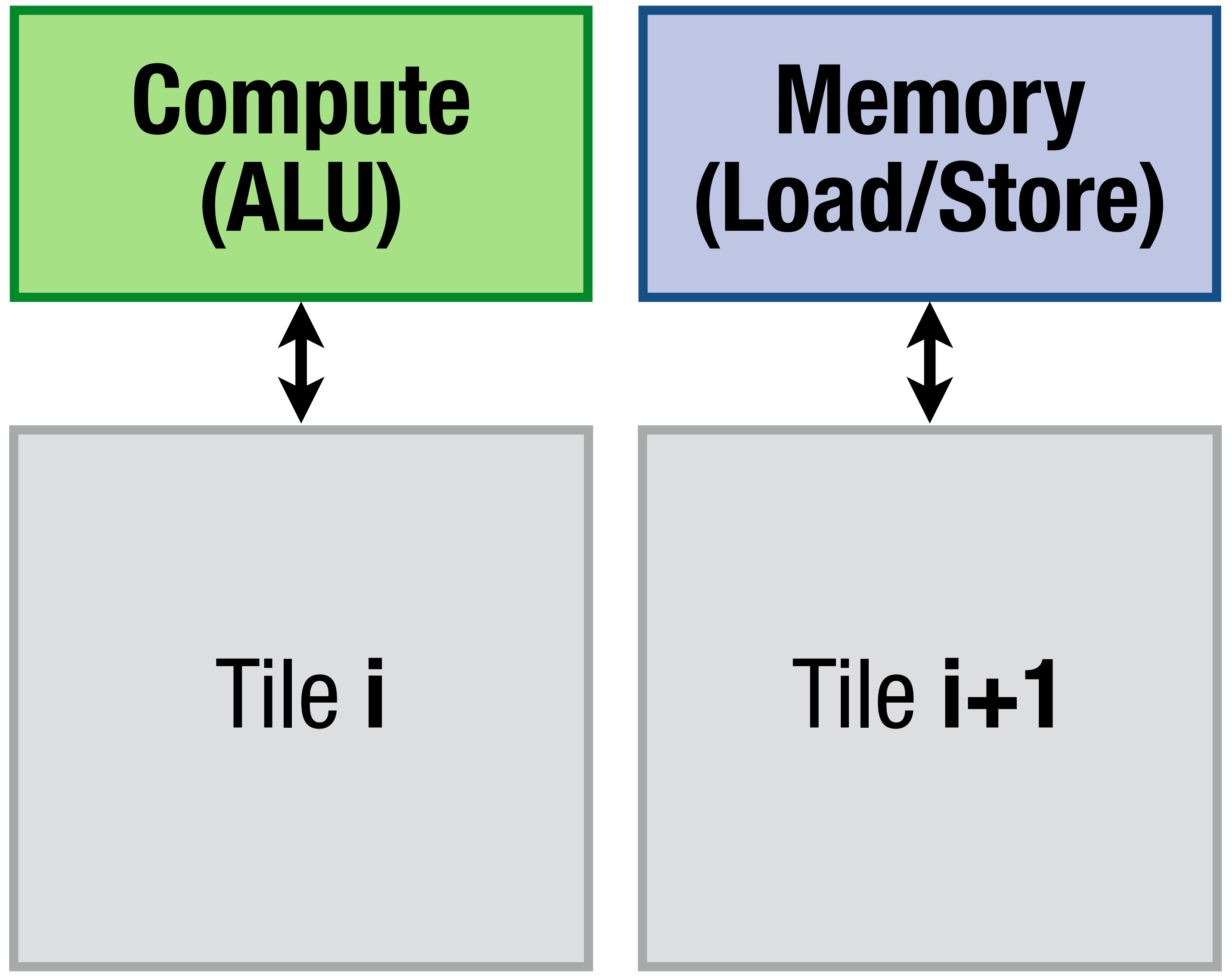


Problem 3: overlapping compute with loading to the **scratchpad**

Implementation approach:
warp specialization

```
foreach tile:
  // load into scratchpad
  for i,j:
    async load next A,B → scratch
  await previous tile load
  // compute!
  for i,j,k:
    compute C += A*B
  sync & swap buffers…
```

Solution:
asynchronous fetch &
double-buffering



Problem 4: load to scratch wastes issue slots, register file space & bandwidth

```
ld.global r1, [G]
st.shared [S], r1
…
ld.shared r2, [S]
```

```
cp.async.shared.global \
 [S], [G], 16
…
cp.async.wait_all
ld.shared r2, [S]
```

Alternative implementation:
async memcpy instructions

Questions?